AD-A203 815

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AI Memo 1094	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	(4)
4. TITLE (and Subtitle): Intelligence in Scientific Computing	5. TYPE OF REPORT & PERIOD COVERED: memorandum
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Hal Abelson, Michael Eisenberg, Mathew Halfant, Jacob Katzenelson, Elisha Sacks, Gerald J. Sussman, Jack Wisdom, Ken Yip	8. CONTRACT OR GRANT NUMBER(s): N00014-86-K-0180
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Artificial Intelligence Laboratory, 545 Technology Square, Cambridge, Massachusetts 02139	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Advanced Research Projects Agency, 1400 Wilson Blvd, Arlington, Virginia 22209	12. REPORT DATE: November 1988
	13. NUMBER OF PAGES: 34
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office): Office of Naval Research, Information Systems, Arlington, Virginia 22217	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this document is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

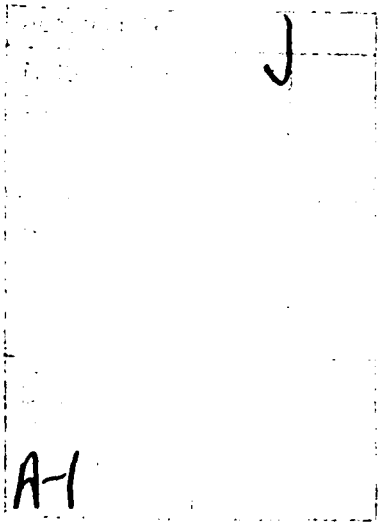
None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Combining numerical techniques with ideas from symbolic computation and with methods incorporating knowledge of science and mathematics leads to a new category of intelligent computational tools for scientists and engineers. These tools autonomously prepare simulation experiments from high-level specifications of physical models. For computationally intensive experiments, they automatically design special-purpose numerical engines optimized to

perform the necessary computations. They actively monitor numerical and physical experiments. They interpret experimental data and formulate numerical results in qualitative terms. They enable their human users to control computational experiments in terms of high-level behavioral descriptions.



MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

AI Memo 1094 *November 1988*

Intelligence in Scientific Computing

Harold Abelson, Michael Eisenberg, Mathew Halfant,
Jacob Katzenelson, Elisha Sacks, Gerald Jay Sussman,
Jack Wisdom, Ken Yip

ABSTRACT

Combining numerical techniques with ideas from symbolic computation and with methods incorporating knowledge of science and mathematics leads to a new category of intelligent computational tools for scientists and engineers. These tools autonomously prepare simulation experiments from high-level specifications of physical models. For computationally intensive experiments, they automatically design special-purpose numerical engines optimized to perform the necessary computations. They actively monitor numerical and physical experiments. They interpret experimental data and formulate numerical results in qualitative terms. They enable their human users to control computational experiments in terms of high-level behavioral descriptions.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of The Department of Defense under Office of Naval Research contract N00014-86-K-0180.

Intelligence in Scientific Computing

Harold Abelson, Michael Eisenberg, Matthew Halfant,
Jacob Katzenelson, Elisha Sacks, Gerald Jay Sussman,
Jack Wisdom, Kenneth Yip

Combining numerical techniques with ideas from symbolic computation and with methods incorporating knowledge of science and mathematics leads to a new category of intelligent computational tools for scientists and engineers. These tools autonomously prepare simulation experiments from high-level specifications of physical models. For computationally intensive experiments, they automatically design special-purpose numerical engines optimized to perform the necessary computations. They actively monitor numerical and physical experiments. They interpret experimental data and formulate numerical results in qualitative terms. They enable their human users to control computational experiments in terms of high-level behavioral descriptions.

As an example of such a tool, imagine an ocean engineer designing a offshore mooring tower for large ships. When standing free, such a tower can be modeled in a straightforward way as an inverted pendulum anchored to the sea bed, driven by wave motion, and restored to vertical position by its buoyancy in sea water. However, a massive ship attached to the mooring structure vastly complicates the dynamics: when the mooring line is slack, the restoring force is the same as before; but when the line is taut, the ship also exerts a force through tension in the line. Numerical simulations of such systems have shown that the effect of even small waves can lead to potentially dangerous resonances that can make the structure fail [25]. The goal of the engineer is to design a structure that can withstand the worst conditions that can be expected to occur in the environment without making the structure excessively expensive.

The following excerpt of a dialogue shows how we imagine an intelligent computer assistant aiding an ocean engineer in the design of such a structure. Assume that the engineer has already proposed a particular structure, either by drawing it or by combining previously defined structures.

⋮

Computer: The dynamical equations for this model are: *(displays a very large set of equations).*

Engineer: Call that the `full model`. Try to find a simpler approximate model.

Computer: The time constants for the bending moments and torsional stresses in equations 1 through 25 are very small compared with the period of lowest mode of the structure seen as a pendulum and with the periods of the largest components in our standard wave spectrum for this region of the gulf coast. Because the deflections of the structure are small we can use the following approximate model: *(displays a much smaller number of equations).*

Engineer: Call that the `simplified model`.

Engineer: Characterize the response of the `simplified model` to the standard wave spectrum.

Computer: State-space analysis shows that the response of this system can be broken into three regions. For very small deflection angles (smaller than 2 degrees) there are stable oscillations, damped with time constants similar to the oscillation frequencies. Significant deflections toward the `boat` show trajectories with competing domains of attraction. There are large subharmonic responses. For deflections larger than about 10 degrees we find complex chaotic motions, but such large deflections violate the assumptions of the model.

Engineer: Characterize the response to the dominant frequency in the spectrum as the amplitude varies from 1 foot to 20 foot seas.

Computer: There are two families of periodic motions here, A and B. Family A is present at the start of the parameter range $s = 1$, with a periodic motion of order 1. At $s = 5.2$ this motion undergoes a symmetry-breaking bifurcation and splits into a pair of motions A_1 and A_2. As s increases, both A_1 and A_2 undergo period-doubling cascades: order 2 ($s = 6.5$), order 4 ($s = 9.8$), order 8 ($s = 10.7$), order 16 ($s = 10.9$) The cascade was not traced passed the order 16 motions. Chaotic motions were observed at $s = 12$. Family B begins with a motion of order 1 arising from a fold bifurcation at $s = 8$. This motion persists through the end of the parameter range $s = 20$.

Engineer: How do the maximum stresses affect the structure?

Computer: Family A motions are within the safe operating region for structures of type UG/751 construction, but the family B motions can easily exceed the safe operating region in a 50-year storm.

Engineer: Perform the same analysis, assuming that we triple the bouyancy of the float.

Computer: The qualitative behavior in the two situations differs slightly in the initial phase of growth of oscillations. Once oscillations are established the behavior is approximately the same. The maximum deflections differ by less than 10% between the two situations, but the frequencies are very different.

Engineer: Begin design of an `active stabilizer` to damp the family B motions. Assume we install thrusters at positions halfway up the structure and at 90% of the height of the structure. Expand the full model to include drives from these thrusters. ...

⋮

No systems as sophisticated as this engineer's assistant yet exist. However, computer programs incorporating important pieces of it are already in place. Generating stylized natural language is not too hard, although understanding unstructured English text is still difficult, and we will not address that problem in this paper. Additionally, our discussion is not really about ocean engineering; the scenario above is not intended to illustrate good design practice in that domain. Rather, our concern here is with the development of intelligent techniques appropriate for the automatic preparation, execution, and control of numerical experiments, and with the automatic interpretation of their results.

- Our envisioned engineer's assistant begins with a description of a mechanism and automatically generates efficient numerical programs that predict its dynamical behavior. This may require more than just straightforward simulation. A stability-analysis task such as "characterize the response to the dominant frequency in the spectrum" requires compiling procedures that evolve, in addition to the state, the variations with respect to changes in initial conditions and the sensitivities with respect to changes in parameters.

- The engineer's assistant automatically prepares high-performance numerical experiments. It has extensive knowledge of numerical methods and it can compose appropriate and correct numerical procedures tailored to the specific application. For critical applications, this compilation can be targeted to the automatic synthesis of special-purpose hardware.

- The engineer's assistant interprets the results of numerical experiments in high-level qualitative terms. This interpretation is based on general mathematical and physical knowledge that constrains the kind of behavior to expect. The interpretation is used to prepare a report to the user, but it is also used in the experimental protocol. The summary of behavior produced from observations of the results of previous experiments is used to automatically select critical values of experimental parameters for subsequent experiments, thus efficiently uncovering the salient phenomena.

Section one of this paper demonstrates significant portions of these capabilities. These include the automatic preparation and monitoring of numerical simulations, the automatic generation of qualitative interpretations of numerical results, and the achievement of breakthrough performance on computationally-demanding problems with the aid of specially-designed computers. (Our special-purpose engine for computing planetary motions has produced the first solid numerical evidence that the solar-system's long-term dynamics is chaotic, thereby answering the famous question of the stability of the solar system.)

Section two takes a closer look at the technology behind these demonstration results. We explain how algorithms from computer vision are applied to interpret phase-space diagrams in dynamics. We illustrate how knowledge about dynamical systems can be encoded using constraints and symbolic rules. We show how to formulate numerical algorithms at appropriate levels of abstraction with higher-order procedures and how to combine these with symbolic algebra to automatically generate numerical programs.

Section three sketches some next steps required to realize the vision of systems like the engineer's assistant.

1 Numerical modeling can be automated

In a typical numerical modeling study, an investigator repeatedly prepares and runs a series of computations and examines the results at each step to select interesting new values for parameters and initial conditions. When enough values have been tried, the investigator classifies and interprets the results. Even with powerful numerical computers, this process requires substantial human effort to prepare simulations, and it relies upon significant human judgment to choose interesting values for parameters, to determine when a simulation run is complete, and to interpret numerical results in qualitative terms.

This section exhibits three programs that automate much of the above process. The *Bifurcation Interpreter* investigates the steady-state orbits in parameterized families of dynamical systems, classifying the types of orbits and the bifurcations through which they change as parameters vary. The *KAM* program autonomously explores nonlinear conservative systems and produces qualitative descriptions of phase-space portraits and bifurcations. Both programs automatically generate summary reports similar to those appearing in published papers in the experimental dynamics literature and in engineering studies of artifacts that have complex dynamics, such as airfoils, ship hulls, and mooring structures. In addition, the capabilities demonstrated by these programs have application in the design of intelligent automatic control systems. The breadth of applicability is illustrated by the *Kineticist's Workbench*, a program that models how chemists understand complex chemical reactions. It combines numerical and symbolic methods to characterize reaction mechanisms in qualitative terms that are useful for the working chemist.

We also discuss the place of special-purpose numerical engines as scientific instruments and survey significant results in planetary dynamics obtained using the *Digital Orrery*.

1.1 Programs can discover and interpret qualitative behavior

In a nonlinear dynamical system with a periodic drive, motion starting from any set of initial conditions will typically evolve to a steady-state orbit. [1] For a parameterized family of dynamical systems, tracing the changes in steady-state orbits as the parameters vary provides a valuable summary of the family's qualitative behavior. Much research in nonlinear dynamics is devoted to studying these *bifurcations*, or changes in type, of steady-state orbits. For one-parameter families at least, the bifurcations generically encountered have been classified and are well-understood. Some examples are the *fold* bifurcation, at which a stable orbit can appear or vanish, the *flip* bifurcation, at which the period of an orbit doubles, and the *pitchfork* bifurcation, at which an orbit splits into two orbits of the same period. There are also commonly-observed bifurcation sequences that occur as the parameter varies. An example is the *period-doubling cascade*, where the order of an orbit successively doubles via a sequence of increasingly closely spaced flip bifurcations, producing chaos.[2]

Dynamicists commonly gain insight into the qualitative behavior of nonlinear systems by developing summary descriptions of steady-state orbits and bifurcations. Figure 1 reproduced here from [7] shows a schematic summary drawn by a physicist based on numerical studies of the two-dimensional Navier-Stokes equation for an incompressible fluid. As the Reynolds number of the fluid increases, the steady-state orbits evolve through a sequence of bifurcations. The diagram summarizes how the evolving orbits can be grouped into four distinct families.

The *Bifurcation Interpreter*, a computer program being developed at MIT by H. Abelson, automatically generates such summary descriptions for one-parameter families of periodically-driven dynamical systems. The dynamical

[1] Possible types of steady-state orbits are periodic orbits, quasi-periodic orbits (which have discrete-frequency spectra, but not at rational multiples of the drive period), and chaotic orbits (which, loosely speaking, are steady-state orbits that are neither periodic nor quasi-periodic).

[2] Various authors use different, and sometimes incompatible terminology to refer to these bifurcation types. For example, the flip is sometimes called a cusp or a pitchfork. We have adopted the terminology used in the book by Thompson and Stewart [26], which provides an introduction to the methods of nonlinear dynamics together with an extensive bibliography.

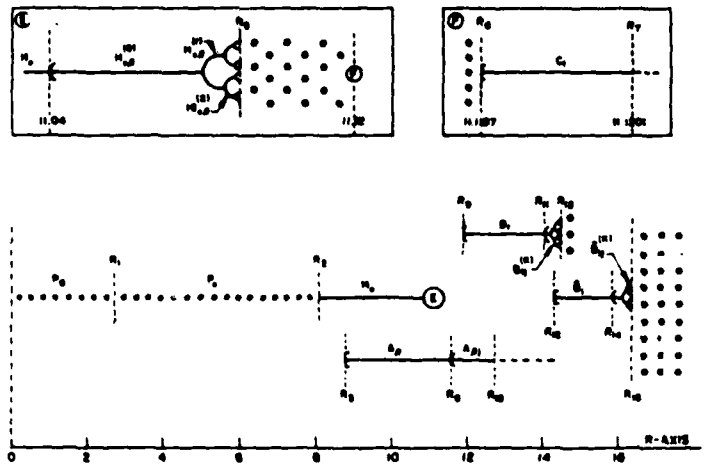


FIG. 16. Graphical summary of the phenomenology of the eight-mode model as R varies. A sequence of dots is used to represent a stable fixed point, a continuous line to represent a stable periodic orbit, a broken line an unstable periodic orbit, a set of stars a strange attractor. A bracket represents a tangent bifurcation, a single parenthesis represents a symmetry breaking bifurcation, and the "pitchfork" a sequence of period-doubling bifurcations. The (100-1) enlargement E, relative to the R interval [11.04,11.12], and the (10 000-1) enlargement F, relative to the R interval [11.1197,11.1201], make visible the phenomenology in these ranges of the parameter.

Figure 1: This diagram, reproduced from a published paper in fluid mechanics, is a physicist's schematic summary description of an approximation to the two-dimensional Navier-Stokes equation for an incompressible fluid. The varying parameter here is the Reynolds number.

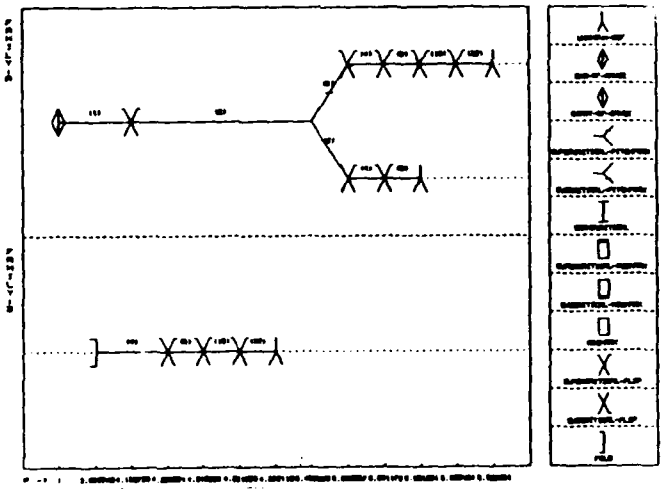


Figure 2: The Bifurcation Interpreter automatically generates summary descriptions of dynamical systems similar to those appearing in published papers. This (preliminary version) diagram describes the behavior of a periodically-impacted hinged beam as the load varies, exhibiting the evolution of two different families of steady-state motions.

system can be specified by differential equations to be integrated, by a period map that directly computes successive states at multiples of the drive period, or by a description of a physical model such as an electrical network. Given a dynamical system, a parameter range to explore, and a domain in state-space, the interpreter discovers periodic orbits, tracks their evolution as the parameter varies, and locates and classifies bifurcations. Using this information, the program categorizes the orbits into families and produces a summary report that describes each family and its evolution through bifurcations.

Here is a sample input to the interpreter:

Period map

$$\begin{array}{lcl} x & \mapsto & x - \dfrac{1-e^{-2m}}{2m} a \sin x + \dfrac{1-e^{-2m}}{2m} y \\ y & \mapsto & -e^{-2m} a \sin x + e^{-2m} y \end{array}$$

Values for fixed parameters

m	0.1π

Range for varying parameter

a	1 to 7

Bounds on state variables

x	$-\pi$ to π (periodic)
y	-4 to 4

The system to be investigated models the vibration of a hinged bar with viscous damping subjected to a fixed-direction periodic impact load at the free end. This problem is discussed in [13], which derives the period map for this motion. Here x is the angular displacement and y is the angular velocity, a specifies magnitude of the load and m specifies the damping factor. The interpreter is asked to explore the system as the load ranges from 1 to 7. With this specification, the interpreter analyzes the system and generates the following report:

> There are 2 distinct families of periodic orbits, A and B.
>
> Family A is already present at the start of the parameter range $a = 1$ as a periodic orbit A_0 of order 1. At $a = 4.130$ there is a supercritical flip bifurcation at which A_0 undergoes period doubling to produce

a periodic orbit A_1 of order 2. At $a = 6.489$ there is a supercritical pitchfork bifurcation at which the family A splits into subfamilies $A(1)$ and $A(2)$, beginning with A_1 splitting into two periodic orbits of order 2. As the parameter a increases, each subfamily undergoes a period-doubling cascade via a sequence of supercritical flip bifurcations to order 4 at $a = 6.838$, order 8 at $a = 6.891$, order 16 at $a = 6.901$, order 32 at $a = 6.903$. The period-doubling cascade was not traced past the order 32 orbit, which apparently period doubles again at $a = 6.904$.

Family B first appears at $a = 3.969$ with an orbit B_0 of order 4 appearing at a fold bifurcation. As the parameter a increases, B undergoes a period-doubling cascade via a sequence of supercritical flip bifurcations to order 8 at $a = 4.239$, order 16 at $a = 4.239$, order 32 at $a = 4.251$. The period-doubling cascade was not traced past the order 32 orbit, which apparently period doubles again at $a = 4.252$.

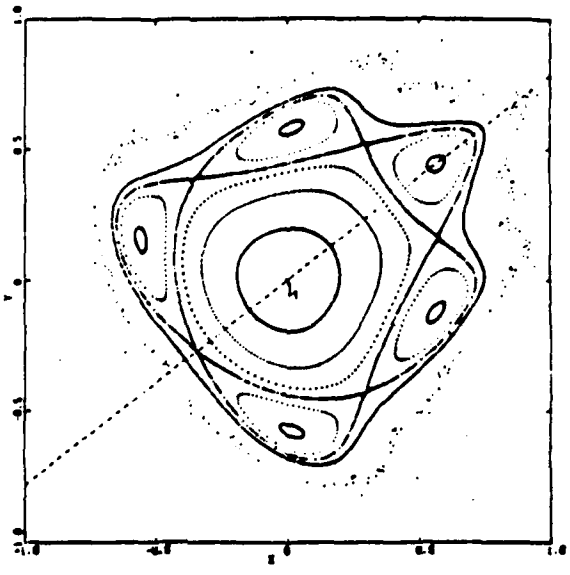
The program can display this same information as a diagram (figure 2) that is similar in style to manually-developed Navier-Stokes analysis in figure 1.[3]

1.2 Smart programs can see what not to compute

Dynamical behavior is complex, but it is not arbitrary. There is structure on phase space that restricts the classes of legal trajectories and provides a grammar of legal phase portraits. For example, trajectories of autonomous systems cannot intersect, and as we vary the initial conditions, the trajectories vary smoothly except at isolated places where the behavior changes. As we vary parameters, the phase portrait changes qualitatively only at bifurcations. In Hamiltonian systems the evolution of the phase space is area-preserving, which greatly restricts the classes of possible structures that can occur in the phase space. This kind of knowledge enables dynamicists to infer a good understanding of a physical system from only a small, but well-chosen, set of experiments.

The phase portrait in figure 3, taken from a historically important paper in dynamics by M. Hénon [9], describes how adding a simple quadratic

[3]The diagram-generation program illustrated in figure 2 was developed by Ognen Nastov.



$$
\begin{aligned}
x &\mapsto x\cos\alpha - (y - x^2)\sin\alpha \\
y &\mapsto x\sin\alpha + (y - x^2)\cos\alpha
\end{aligned}
$$

Figure 3: This phase portrait is Hénon's summary of the dynamics of the map for $\cos\alpha = .24$.

nonlinearity to a linear rotation can lead to dramatic changes in dynamical behavior. Observe how the figure characterizes the dynamics by showing only a few orbits. Presumably, Hénon was able to generate this figure after performing only a few judiciously chosen numerical experiments.

The *KAM* program developed by K. Yip at MIT can analyze systems in the same way [29, 30]. It knows enough about the constraints on the structure of phase space to choose initial conditions and parameters as cleverly as an expert dynamicist. KAM's summary description of Hénon's map is shown in figure 4. Observe that this is almost identical to the summary presented by Hénon. Moreover, KAM was able to deduce this description after trying only ten initial conditions.

KAM's ability to control numerical experiments arises from the fact that it not only produces pictures for us to see—it also *looks at* the pictures it draws, visually recognizing and classifying different orbit types as they numerically evolve. By combining techniques from computer vision with sophisticated dynamical invariants, KAM is able to exploit mathematical knowledge, represented in terms of a "grammar" that dictates consistency

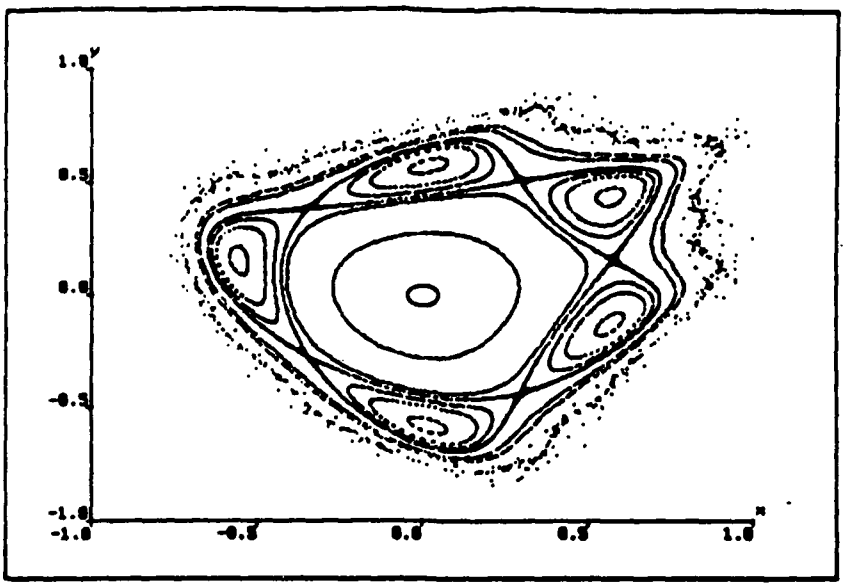


Figure 4: The KAM program generates this summary picture of Hénon's map.

constraints on the structure of phase space. When it chooses new initial conditions to explore, it does so in an attempt to make the picture consistent with these constraints. In addition to drawing the picture, KAM generates a textual analysis that explains what the program "sees." Here is KAM's description of the picture it generates for Hénon's map.

> The portrait has an elliptic fixed point at (0,0). Surrounding the fixed point is a regular region bounded by a KAM curve with rotation number between 1/5 and 1/4. Outside the regular region lies a chain of 5 islands. The island chain is bounded by a KAM curve with rotation number between 4/21 and 5/26. The outermost region is occupied by chaotic orbits that eventually escape.

1.3 Programs can construct and analyze approximations

A powerful strategy for analyzing a complicated dynamical system is to approximate it with a simpler system, analyze the approximation, and map the results back to the original system. The approximations must be accurate enough to reproduce the essential properties of the original system, yet simple enough to be analyzed efficiently. Human experts have found that piecewise linear approximations satisfy both criteria for a wide class of models. The PLR program, developed by E. Sacks at MIT, exploits this fact to

automate the analysis of second-order autonomous ordinary differential equations [21, 22]. It derives the qualitative behavior of intractable equations by approximating them with piecewise linear equations and constructing phase diagrams of the approximations.

PLR constructs a composite phase diagram for a piecewise-linear system by combining the local phase diagrams of its linear regions. It employs the standard theory of linear equations to ascertain the local phase diagrams. Linear systems have simple well-understood dynamics. Either all trajectories are periodic, all approach a fixed point, or all approach infinity. PLR pastes together the local phase diagrams by determining which sequences of regions trajectories can traverse. It summarizes the results by a *transition graph* whose nodes and links represent regions and transitions. Each path through the transition graph of a piecewise-linear system indicates that trajectories traverse the corresponding regions in the prescribed order. Loops denote trajectories that remain in one region forever, whereas longer cycles denote trajectories that continually shift between a sequence of regions.

As a simple example, PLR can qualitatively analyze the behavior of an *undriven van der Pol oscillator*, a *simple nonlinear circuit consisting* of a capacitor, an inductor, and a nonlinear resistor connected in series. The current through the circuit obeys the equation

$$i'' + \frac{k}{L}(i^2 - 1)i' + \frac{1}{LC}i = 0, \tag{1}$$

with C the capacitance, L the inductance, and k a scaling factor. PLR approximates this equation with a piecewise linear equation and constructs the phase diagram and transition graph shown in Figure 5. It deduces that the system oscillates from the fact that tracing edges starting from any node in the graph leads to a cycle. Intuitively, the system oscillates because the nonlinear resistor adds energy to the circuit at low currents and drains energy at high currents.

1.4 Domain knowledge can guide numerical modeling

M. Eisenberg's *Kineticist's Workbench*, also being developed at MIT, is a program that combines general knowledge of dynamics with specific knowledge about chemical reactions in the analysis, understanding, and simulation of complex chemical reaction mechanisms.

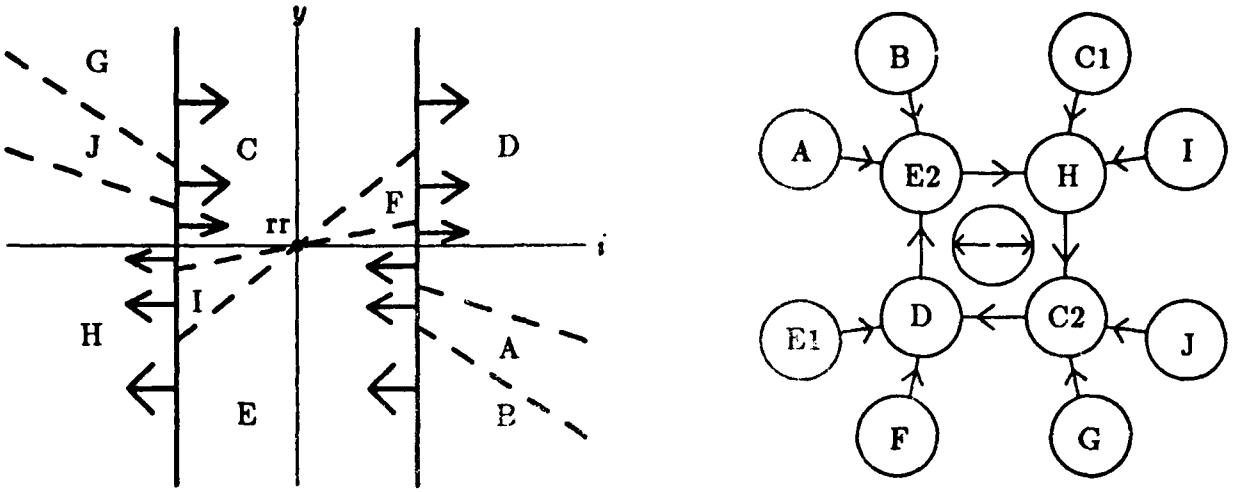


Figure 5: PLR's phase diagram and transition graph for its piecewise linear van der Pol approximation. Arrows indicate boundaries that trajectories cross.

Chemists, in trying to model reactions, typically hypothesize a set of elementary reaction steps (corresponding to molecular collisions) that constitute a proposed pathway for the overall reaction. This collection of elementary steps may be large. It usually gives rise to a mathematical model consisting of many tightly-coupled nonlinear differential equations. The problem of simulating such a system can be formidable, but a simulation merely provides numerical results. Even more important to the chemist is to achieve some sort of *qualitative* understanding of the reaction mechanism. The Kineticist's Workbench combines numerical simulation with a variety of symbolic methods in order to characterize reaction mechanisms in terms that are meaningful to the working chemist such as "equilibrium," "steady states," and "oscillations."

The Workbench also suggests ways to simplify complex mechanisms. The key idea here is that the program can use its qualitative characterization of a given mechanism's behavior to determine which steps within the mechanism have an important behavioral effect, and which are superfluous and may therefore be dropped from the chemist's model.

The following scenario shows how the Workbench is intended to interact with a user in the analysis of a chemical process. The initial information is

specified as follows:

Step no.	Step	Rate Constant
(1)	$C \rightarrow B$	300
(2)	$B \rightarrow C$	100
(3)	$A \rightarrow X$	1
(4)	$B + X \rightarrow Y + D$	1
(5)	$2X + Y \rightarrow 3X$	0.333
(6)	$X \rightarrow E$	1
(7)	$C \rightarrow F$	100

External sources for species C (rate of addition: 100 mmol/liter-sec)

Concentration of A assumed constant at 1 mmol/liter. Initial concentrations of all species besides A are 0 mmol/liter.

With this specification, the Workbench responds:

The mechanism may be decomposed into two submechanisms. The first may be written as follows:[4]

$$(1.1) \quad [External] \leftrightarrow C \leftrightarrow B \rightarrow [External]$$

This submechanism results in a rapid equilibrium between C and B; the final concentrations of both C and B (1 and 3 mmol/liter, respectively) are independent of their initial concentrations.[5]

The second submechanism may be written as follows:

$$(2.1) \quad [External] \leftrightarrow X$$
$$(2.2) \quad B + X \rightarrow Y$$
$$(2.3) \quad 2X + Y \rightarrow 3X$$

[4]The notation $[External]$ used in this example is a catchall used to indicate the presence of external sources or sinks for some species. The notation corresponds roughly to the "zero complex" notation of Feinberg [6].

[5]In performing this simplification, the Workbench has treated the original reaction (7) as the presence of an external sink for species C, since the concentration of species F merely grows with time and does not affect any other species. Similarly, reaction (4), being slow compared to all the reactions included in (1.1), may be treated as a slow (but time-varying) sink for species B. This reasoning could be exposed at the user's request.

> Since the concentration of B in the fi. ' submechanism is constant at 3 mmol/liter, the second submechanism may be further simplified as follows:
>
> $$(2.1') \quad [External] \leftrightarrow X - Y$$
> $$(2.2') \quad 2X + Y \rightarrow 3X$$
>
> Simulation of this second mechanism reveals that the concentrations of X and Y oscillate over time. Each oscillation consists of five phases: (1) slow growth in the concentrations of X and Y; (2) more rapid growth in X; (3) large jump in X and decline in Y; (4) large decline in X and increase in Y; (5) slower decline in X and slow increase in Y. The time between oscillation peaks is approximately 17.8 seconds.

The chemist now asks the program to compare the results of the approximate mechanism (2.1′–2.2′) with the numerically exact results of mechanism (1–7), comparing the qualitative behavior of the two mechanisms as reflected in species X and Y. The workbench responds:

> The qualitative behavior of the two mechanisms differs slightly during the initial phase of slow growth in X and Y. Once oscillations begin, the exact mechanism (1–7) displays the same qualitative structure as the approximate mechanism (2.1′–2.2′), and the period is still approximately 17.8 seconds. The concentrations of X and Y at comparable points in the oscillations differ by less than 2 percent between the two mechanisms.

The chemist now tries a different variation on the original reaction, and alters the original mechanism so that the concentration of A is not constant over time, but is rather an explicit function of time, as given by the equation $[A] = 1 + \sin t$. The workbench responds:

> The decomposition into submechanisms is not affected; nor is the constant concentration of B. Again, the concentrations of X and Y appear to be oscillating, but each oscillation now consists of eight phases, instead of five, as before.

It is worth noting some of the key features of this example. First, the Workbench is able to decompose the original mechanism into two submechanisms, each of which is capable of independent simulation; this simplifies

both the analysis and simulation of the larger mechanism. Second, the Workbench is able to decompose the first of the two submechanisms in terms of a dichotomy between fast and slow steps; this allows the program to approximate the submechanism as a system in equilibrium. Third, the program uses numerical simulation to derive equilibrium concentrations for this submechanism. Finally, the Workbench is able to describe the results of simulating the second submechanism in terms of a succession of qualitative episodes characterized by changing growth rates of the species X and Y.

1.5 Fast computers need not be large or expensive

Numerical modeling often requires substantial resources. Scientists and engineers have traditionally obtained these resources either by acquiring large-scale computers or renting time on them. However, a specialized computer can be simple and physically small. Indeed, it may be just as easy to design, build, and program a special-purpose computer than to develop software for general-purpose supercomputers. Moreover, the specialized computer can become an ordinary experimental instrument belonging to the research group that made it, thus avoiding the administrative burden and the scheduling problems associated with expensive, shared resources.

The question of the stability of the solar system is probably the most famous longstanding problem in astrodynamics. In fact, it was investigations into precisely this problem that inspired Poincaré to develop the modern qualitative theory of dynamical systems. In 1988, G. Sussman and J. Wisdom completed a series of numerical experiments at MIT demonstrating that the long-term motion of the planet Pluto, and by implication the dynamics of the Solar System, is chaotic [24].

The stability question was settled using the Digital Orrery [4], a special-purpose numerical engine optimized for high-precision numerical integrations of the equations of motion of small numbers of gravitationally interacting bodies. Using 1980 technology, the device is about 1 cubic foot of electronics, dissipating 150 watts. On the problem it was designed to solve, it is measured to be 60 times faster than a VAX 11/780 with FPA, or 1/3 the speed of a Cray 1.

Figure 6 shows the exponential divergence of nearby Pluto trajectories over 400 million years. This data is taken from an 845-million-year integration performed with the Orrery. Before the Orrery, high-precision integra-

tions over simulated times of millions of years were prohibitively expensive. The longest previous integration of the outer planets was for five million years, performed on a Japanese supercomputer in 1984 [14]. Even though the Orrery is not as fast as the fastest supercomputer, its small scale and relative low cost mean that it can be dedicated to long computations in ways that a conventional supercomputer could not. To perform the integration that established Pluto's chaotic behavior, the Orrery ran continually for five months.

The Orrery was designed and built by six people in only nine months. This was possible only because of novel software support for the design process. The simulator for the Orrery is partially symbolic—simulated registers hold symbolic values and simulated arithmetic parts combine these to produce algebraic expressions (in addition to checking timing and electrical constraints). This means that a successful simulation yields a simulated memory containing algebraic expressions that can be checked for correctness.

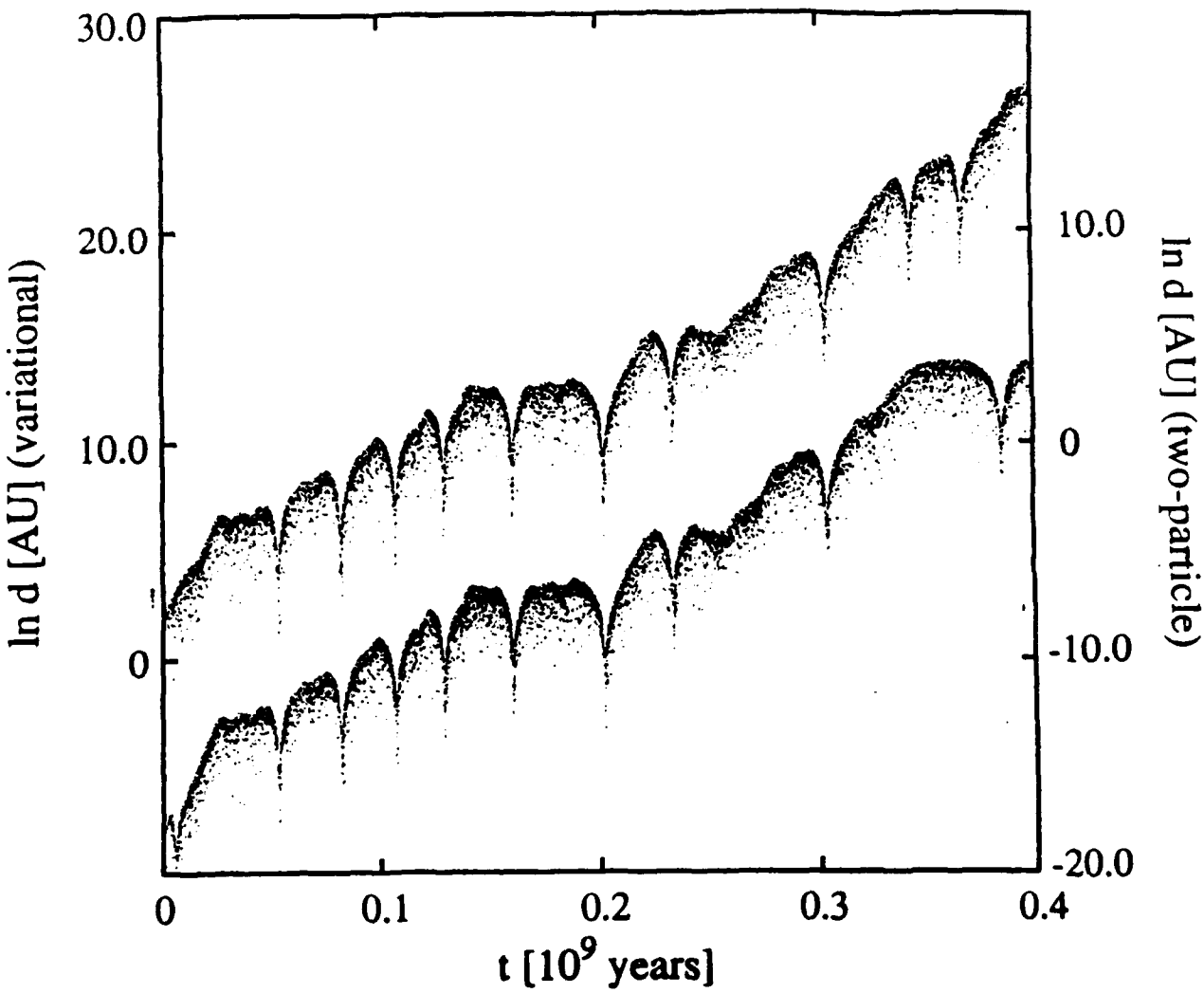


Figure 6: The exponential divergence of nearby trajectories is indicated by the average linear growth of the logarithms of the distance measures as a function of time. In the upper trace we see the growth of the variational distance around a reference trajectory. In the lower trace we see how two Plutos diverge with time. The distance saturates near 45AU; note that the semimajor axis of Pluto's orbit is about 40AU. The variational method of studying neighboring trajectories does not have the problem of saturation. Note that the two methods are in excellent agreement until the two-trajectory method has nearly saturated.

2 Intelligent numerical computing rests on AI technology

The illustrations above achieve their impressive results by bringing symbolic methods to bear on the problems of numerical computation. Some of these techniques are traditional AI methods, which achieve new power when they are combined with deep knowledge of dynamical systems. The KAM program, for example, uses techniques from machine vision to recognize and classify the relevant geometrical properties of the trajectories. The Bifurcation Interpreter uses algebraic manipulation and knowledge about the local geometry of bifurcations to automatically generate numerical procedures that track periodic orbits. The key to automatically generating high-performance numerical algorithms is to express knowledge of numerical analysis at an appropriate level of abstraction. This is supported by a library of numerical methods that is organized around the liberal use of higher-order procedural abstractions. With this organization, one constructs sophisticated numerical methods by mixing and matching standard components in well-understood ways. The resulting programs are both more perspicuous and more robust than conventional numerical methods. For example, a procedure by G. Roylance that automatically generates special functions has constucted a Bessel-function routine that is 40 times more accurate than the National Bureau of Standards approximation, for the same amount of computation.

2.1 The KAM program exploits techniques from computer vision

Yip's KAM program is notable because it applies judgement, similar to that of an expert dynamicist, in directing the course of its numerical experiments. In making judicious choices of what to try next, KAM must interpret what it sees. This process occurs in three phases: aggregation, clustering, and classification. The images of an initial point produced by iterating the map forms a set of isolated points. This orbit must be classified. In Hamiltonian systems there are three types of orbits to distinguish. In a surface of section, periodic orbits appear as isolated points, quasiperiodic orbits appear as closed curves or island chains, and chaotic orbits appear to take up regions of 2-dimensional space. KAM must also aggregate the components of an orbit so

that it can be further classified. It must be able to determine the number of islands in an island chain, as this number gives the period of the enclosed periodic point. KAM must be able to estimate the centroid and area enclosed by a curve and to recognize the shape of a curve. KAM implements these abilities with techniques from computational geometry and computer vision.

KAM classifies orbits using methods based on the Euclidean minimal spanning tree—the tree that interconnects all the points with minimal total edge length—which it constructs by means of the Prim-Dijkstra algorithm [5]. For each sub-tree of the spanning tree, KAM examines the degree of each of its nodes, where the degree of a node is the number of nodes connected to it in the sub-tree. For a smooth curve, the spanning tree consists of two terminal nodes of degree one and other nodes of degree two. For a point set that fills an area, its corresponding spanning tree consists of many nodes having degree three or higher (figure 7).

To aggregate points, KAM deletes from the tree edges that are significantly longer than nearby edges, following an aggregation algorithm suggested by Zahn [31]. This divides the tree into connected components. Figure 8 shows how the program aggregates points of a quasiperiodic orbit and recognizes it as an island chain.

To compute the area and centroid of the region bounded by a curve, KAM generates an ordered sequence of points from the spanning tree, and spline-interpolates the sequence to obtain a smooth curve. Straightforward algorithms are then applied to compute the area and centroid. Shape recognition is accomplished using scale-space methods pioneered by Witkin [28].

2.2 AI techniques can implement deep mathematical knowledge

Viewed as abstract examples of AI technology, our demonstration programs are hardly novel. The uniqueness of these programs, and the source of their power, is that they use classic AI methods to exploit specific domain knowledge based on rigorous mathematical results.

PLR for instance, combines geometric reasoning, symbolic algebra, and inequality reasoning to test whether trajectories of a piecewise linear system cross between adjacent regions in phase space. For a trajectory to cross from region R to S via boundary u, its tangent t at the intersection point with

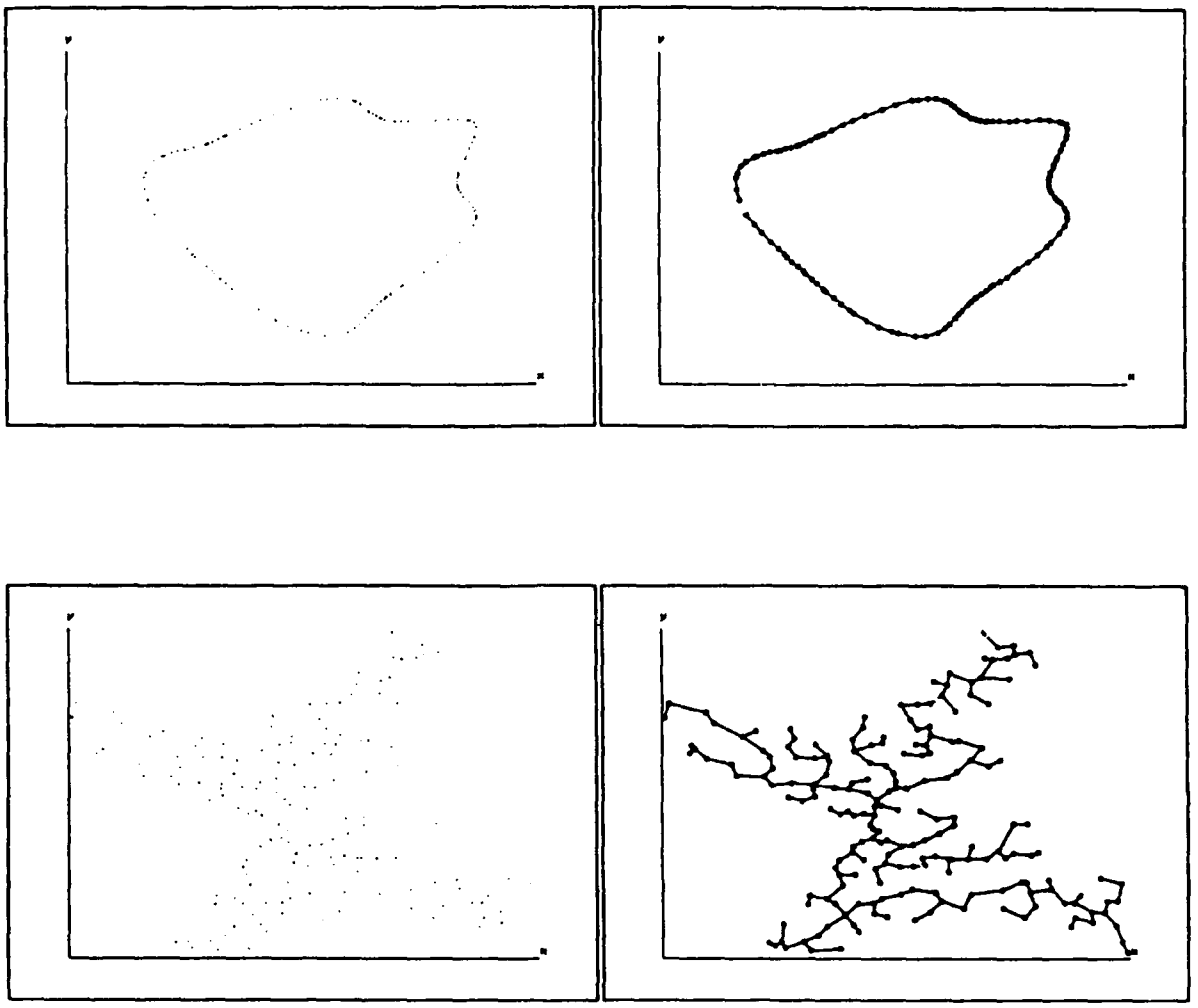


Figure 7: Starting with the successive iterates of a point, KAM classifies orbits using algorithms from machine vision. As shown above, a quasiperiodic orbit can be distinguished from a chaotic orbit by examining the branching factor in the Euclidean minimal spanning tree.

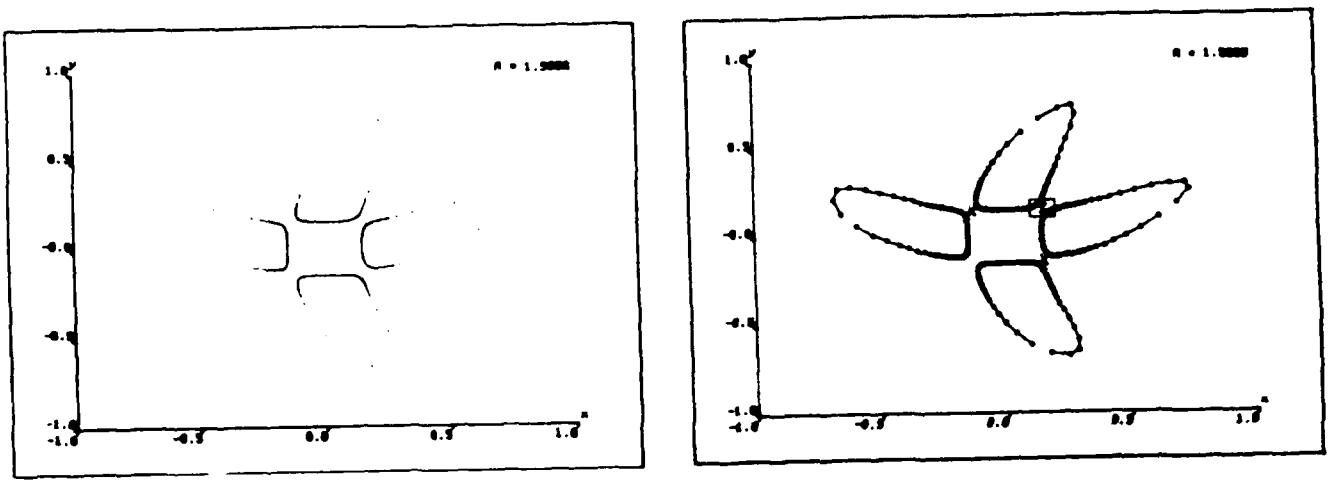


Figure 8: KAM uses the minimal spanning tree to cluster orbits into components. The components of an island chain can be isolated by detecting long edges in the spanning tree and deleting these from the graph.

u must form an acute angle with the normal n, as shown in Figure 9. This geometric condition is equivalent to the algebraic condition that the inner product $t \cdot n$ be positive. Hence, a transition exists from R to S unless $t \cdot n \leq 0$ everywhere on u. PLR resolves the inequality $t \cdot n \leq 0$ on u with the **BOUNDER** inequality reasoner [20].

PLR combines symbolic reasoning with deep knowledge about dynamical systems to interpret the transition graphs that it constructs. For example, the transition graph for the van der Pol equation shows that all trajectories spiral around the origin, but tells nothing of whether they move inward, move outward, or wobble around. PLR invokes a difficult theorem to prove that all trajectories converge to a unique limit cycle. It tests the preconditions of the theorem by proving inequalities and manipulating symbolic expressions.

The KAM program limits the number of phase-space trajectories it must explore by drawing upon constraint analysis, as pioneered by Waltz [27]. As in any constraint analysis, KAM relies upon "grammar" that expresses the consistent ways in which primitive elements can be combined. In KAM's case, the primitive elements incorporate sophisticated mathematical invariants,

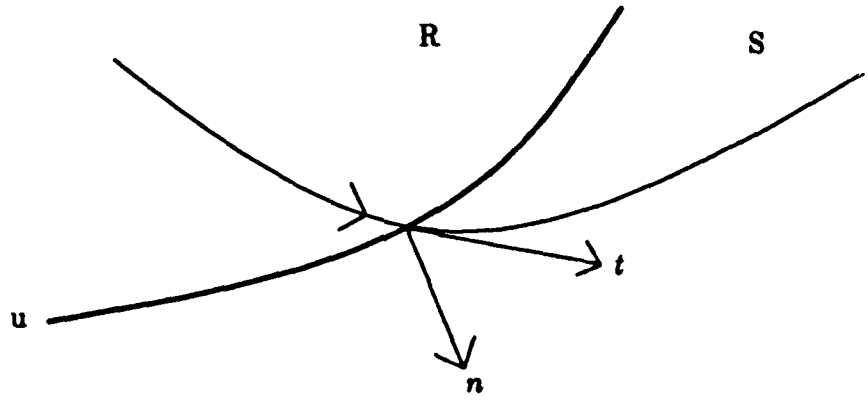


Figure 9: Trajectory crossing from R to S via u: the tangent t at the crossing point must form an acute angle with n, the normal to u that points into S.

and the grammatical rules embody deep theorems about the behavior of dynamical systems.

One such invariant, for example, is the *rotation number* of an orbit, a quantity that measures the asymptotic average of the angular distances between any two successive iterates in units of 2π-radians. A rule in KAM's grammar embodies a theorem that the rotation numbers of nearby orbits must change continuously [16]. As an example of how this is used, suppose that KAM has located two nearby almost-periodic orbits having rotation numbers ρ_1 and ρ_2 respectively. Suppose ρ_1 is slightly smaller than $1/5$, and ρ_2 slightly larger. With only these two orbits, KAM's evolving phase-space picture cannot be complete. By continuity, KAM expects to find a third, nearby orbit with rotation number exactly equal to $1/5$, that is, a periodic orbit of period 5, which KAM proceeds to search for and classify.

In a similar manner, Abelson's Bifurcation Interpreter draws upon knowledge of the geometry of typical changes in the steady-state orbits of one-parameter families of dynamical systems. Periodic orbits of a periodically-driven oscillator can be identified as fixed-points of the *period map*, which maps a state to the end-point of the trajectory starting from that state and evolving for one period of the drive. Stability of an orbit is determined by the stability of the corresponding fixed point. If the interpreter notices that a stable orbit suddenly becomes unstable as the family parameter increases, it attempts to explain this change as the result of a bifurcation. The type of bifurcation can be conjectured by examining the eigenvalues of the period

map at the fixed point, and the conjecture can be verified by a search that is tailored to the local geometry for that bifurcation type.

For example, for a stable orbit, the complex eigenvalues of the corresponding fixed point of the period map must lie within the unit circle. If stability is lost with an eigenvalue apparently crossing the unit circle at -1, a classification theorem for bifurcations [11] tells the interpreter to expect that this is a supercritical-flip bifurcation, which corresponds to a period doubling of the orbit. Near the bifurcation one should expect to see a stable orbit just before the critical parameter value and, just after the critical value, an unstable orbit together with a stable orbit of double the period. For this type of bifurcation, the interpreter attempts to locate the new expected orbits using a search technique that detects fixed points by computing the *Poincaré index* of the period map [12]. If these orbits are located, the bifurcation is probably a flip. If a different local geometry is found, the apparent bifurcation may be the result of numerical error, or the interaction of two nearby bifurcations, or it may be a bifurcation of non-standard type. In any case, the result of the search is passed to a critic that attempts to reconcile the local results for all bifurcations detected and produce a consistent description.

Going beyond general knowledge of dynamical systems, the Kineticist's Workbench program employs a number of techniques specific to the domain of chemical kinetics. For example, the program examines the qualitative history of a reaction simulation, attempting to find periods of time during which concentrations of some species may be treated as constant; this is an automation of the kind of steady-state analysis that is a staple of kinetic investigation [15]. Another portion of the program—that portion devoted to spotting fast equilibria within a mechanism—makes extensive use of the *reaction network* formalism developed by Feinberg, Horn, Jackson, and coworkers at the University of Rochester [6]. An especially fruitful result of their work is the *zero-deficiency theorem*, which provides a simple algorithmic test for determining whether a reaction mechanism gives rise to stable equilibria. Finally, the portion of the Workbench program devoted to decomposing mechanisms according to mutual influence between species may be used to identify transient species, and to drop those species from a particular simulation as soon as their concentrations are deemed too low to affect the remainder of the simulation.

```
(define-network driven-van-der-pol
  ((a parameter v/i^3)
   (b parameter resistance)
   (d drive voltage))
  (n1 n2 n3)
  (parts
   (nl-res non-linear-resistor (n+ n3) (n- gnd)
           (vic (lambda (v i)
                  (= v (- (* a i i i) (* b i))))))
   (l inductor (n+ n1) (n- n2))
   (c capacitor (n+ n2) (n- n3))
   (s voltage-source (n+ n1) (n- gnd) (strength d))))
```

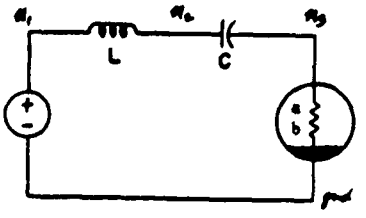


Figure 10: The wiring diagram of a simple nonlinear circuit is described by means of a special-purpose language of electrical networks. This description can be automatically compiled into numerical procedures that evolve the state of the system.

2.3 Numerical experiments can be prepared automatically

Translating high-level specifications into high-quality numerical routines can be a tedious and error-prone process whose difficulty limits the utility of even the most powerful numerical computers. A program by H. Abelson and G.J. Sussman draws upon a spectrum of computational tools—numerical methods, symbolic algebra, and semantic constraints (such as dimensions)—to automate the preparation and execution of numerical simulations [2]. These tools are designed so that combined methods, tailored to particular problems, can be constructed on the fly. One can use symbolic algebra to automatically generate numerical procedures, and one can use domain-specific constraints to guide algebraic derivations and to avoid complexity.

Figure 10 shows the wiring diagram for a simple nonlinear circuit, a driven van der Pol oscillator consisting of voltage source, a capacitor, an inductor, and a nonlinear resistor with the cube-law characteristic $v = ai^3 - bi$. The figure also shows a description of this wiring diagram in a language formulated for describing electrical networks. This description specifies circuit's parameters, its primitive parts, and how the parts are interconnected.

Given this description, the program combines models of the primitive

elements to form equations that are then algebraically solved to produce state equations for the van der Pol oscillator. The state equations are compiled into a procedure (the system derivative) that will evolve the system numerically when combined with an appropriate numerical integrator.

These operations can involve a nontrivial amount of algebraic manipulation. Even for systems that are specified in closed form, most nonlinear systems cannot be algebraically solved to produce explicit state equations. In the general case, the program recognizes variables that cannot be eliminated from the state equations and compiles an iterative scheme for approximating these variables. This requires symbolic differentiation to produce a Jacobian that is incorporated into a Newton-Raphson search and to augment the system state so that it will evolve good starting points for Newton's method at each step of the integration.

For applications such as the Bifurcation Interpreter, one must also compile numerical routines that find period orbits and track them as the system parameters vary. Finding and tracking periodic orbits rests upon determining the sensitivity of trajectories to variations in initial state and parameters. This can be done by evolving the variational system, obtained by symbolic manipulation of the state equations, and by evolving various derived systems obtained by differentiating the state equations with respect to parameters. Figure 11 shows a numerical routine, automatically generated from the circuit description in figure 10, that can be combined with a numerical integrator to evolve the states and the variational states of the driven van der Pol oscillator.

The result of the physical modeling, algebraic manipulation, and fancy compilation shown above in figure 11 is a higher-order procedure—a *system-derivative generator* for the dynamical system under study. The generator takes as arguments numerical values for the system parameters and produces a *system-derivative* procedure, which takes a system state vector as argument and produces a differential state (a vector that when multiplied by an increment of time is an increment of state). This system-derivative procedure is passed to an *integration driver* that returns a procedure which, given an initial state, evolves the system numerically.

Since all system derivative procedures are constructed to respect the same conventional interfaces, we may choose from a variety of integration methods. Moreover, the integration methods themselves can be automatically constructed from a library of procedures that can be used as interchangeable

```
(lambda (c.c l.l d b a)
  (lambda (*varstate*)
    (let ((t (vector-ref *varstate* 0))
          (v.c (vector-ref *varstate* 1))
          (i.l (vector-ref *varstate* 2))
          (v.c.del.v.c (vector-ref *varstate* 3))
          (v.c.del.i.l (vector-ref *varstate* 4))
          (i.l.del.v.c (vector-ref *varstate* 5))
          (i.l.del.i.l (vector-ref *varstate* 6)))
      (let ((g27 (* a i.l i.l)))
        (let ((g28 (* -3 g27)))
          (vector 1
                  (/ i.l c.c)
                  (/ (+ (* -1 g27 i.l) (* -1 v.c) (* b i.l) (d t))
                     l.l)
                  (/ v.c.del.i.l c.c)
                  (/ (+ (* -1 v.c.del.v.c)
                        (* b v.c.del.i.l)
                        (* g28 v.c.del.i.l))
                     l.l)
                  (/ i.l.del.i.l c.c)
                  (/ (+ (* -1 i.l.del.v.c)
                        (* b i.l.del.i.l)
                        (* g28 i.l.del.i.l))
                     l.l)))))))
```

Figure 11: This numerical procedure is the augmented system derivative generator for evolving variational states for the driven van der Pol oscillator. The procedure was automatically generated from the circuit description shown above.

components in the construction of traditional applications. Going further, we believe that it is not difficult to automatically implement these numerical procedures as special-purpose hardware, like the Orrery.

To support the automatic construction of numerical procedures, we are developing a kernel numerical library that is organized around the liberal use of high-order procedural abstractions. For example, figure 12 illustrates this mix-and-match construction of numerical routines, expressing Romberg's method of quadrature as a combination of trapezoidal integration and Richardson extrapolation, following the exposition given by M. Halfant and G.J. Sussman in [8]. Such a formulation is valuable in that it separates the ideas into several independent pieces. Clever ideas need be coded and debugged only once, in a context independent of the particular application, thus enhancing the reliability of software built in this way. For instance, G. Roylance [19] shows how to construct high-performance implementations of special functions, abstracting recurrent themes such as Chebyshev economization. His automatically-constructed procedure for computing Bessel functions is 40 times more accurate, for the same number of terms, than the approximation specified in the National Bureau of Standards tables [3]. More significantly, Roylance's formulation clearly exposes the underlying approximation methods so that parameters, such as the required precision of the routines, can be changed at will.

Besides providing a convenient target for automatic construction of numerical procedures, powerful abstraction mechanisms help us to express some of the vocabulary and methods of numerical analysis in a form that is close to the mathematical theory, and is thus easy to understand and check. A program is a communication, not just between programmers and computers, but also between programmers and human readers of the program; quite often, between the programmer and him/herself. One power of programming is that it allows one to make the knowledge of methods explicit, so that methods can be studied as theoretical entities. Traditional numerical programs are hand-crafted for each application. The traditional style does not admit such explicit decomposition and naming of methods, thus forfeiting much of the power and joy of programming.

```
(define (romberg f a b tolerance)
  (stream-limit
    (richardson-sequence (trapezoid-sums f a b)
                         2
                         2)
    tolerance))

(define (trapezoid-sums f a b)
  (define (next-S S n)
    (let* ((h (/ (- b a) 2 n))
           (fx (lambda(i) (f (+ a (* (+ i i -1) h))))))
      (+ (/ S 2) (* h (sigma fx 1 n)))))
  (define (S-and-n-stream S n)
    (cons-stream (list S n)
                 (S-and-n-stream (next-S S n) (* n 2))))
  (let* ((h (- b a))
         (S (* (/ h 2) (+ (f a) (f b)))))
    (map-stream car (S-and-n-stream S 1))))


(define (richardson-sequence seq start-index inc-index)
  (define (sequences seq order)
    (cons-stream seq
     (sequences
      (let* ((2^p (expt 2 order)) (2^p-1 (- 2^p 1)))
        (map-streams (lambda (Rh Rh/2)
                       (/ (- (* 2^p Rh/2) Rh) 2^p-1))
                     seq
                     (tail seq)))
      (+ order inc-index))))
  (map-streams head (sequences seq start-index)))
```

Figure 12: Romberg's method of quadrature can be built by combining a primitive trapezoidal integrator with an accelerator that speeds convergence of sequences by Richardson extrapolation. The result is an infinite sequence (stream) of increasingly accurate approximations to the definite integral. The same Richardson accelerator can be combined with other sequence generators to build other classical numerical routines.

3 Intelligent tools are feasible

The work described in the preceding sections demonstrates much of the technology required to produce programs that can serve scientists and engineers as intelligent problem-solving partners, programs such as the engineer's assistant that we envisioned at the beginning of this paper.

We have shown how to use symbolic algebra to compile high-level descriptions such as circuit diagrams directly into numerical modeling and simulation programs whose elements can be automatically generated from a library of mix-and-match numerical subroutines expressed at appropriate levels of abstraction. Our experience with the Digital Orrery proves that such numerical programs can be run at supercomputer speeds, without the cost of a general-purpose supercomputer. The Bifurcation Interpreter, KAM and PLR demonstrate that intelligent programs incorporating knowledge of dynamical systems can automatically control and monitor numerical experiments and interpret the results in qualitative terms. The Kineticist's Workbench illustrates how these capabilities can be combined with knowledge about a particular domain to produce a sophisticated tool for modeling and analysis.

3.1 Higher-dimensional systems are hard

Most systems of interest have more than two degrees of freedom, yet the KAM and PLR programs and the Bifurcation Interpreter depend upon special properties of low-dimensional systems. The grammar of possible phase portraits and the catalog of generic bifurcations embodied in these programs cannot easily be extended to higher dimensional systems. On the other hand, there are qualitative features of such systems that can be usefully extracted and used to guide numerical experiments.

Exploring the qualitative behavior of high-dimensional systems requires a combination of analytic and numeric methods. Analytic methods can provide clear definitive information, but are often hard to apply or unavailable. There are well-established method for deriving the local behavior of trajectories in the neighborhood of fixed points, but few tools exist for determining global behavior. On the other hand, a program could detect a saddle analytically, calculate its stable and unstable manifolds numerically, determine whether the manifolds intersect each other, and draw conclusions about global behavior.

Moreover, even in high-dimensional spaces, it is still possible to use clustering techniques to examine the set of iterates of a map or the flow of a differential equation and determine if a trajectory is confined to a lower dimensional submanifold of the formal state space. Each reduction in dimension is evidence of an integral of motion, such as conservation of energy. One can also (as people do) apply visual recognition techniques to low-dimensional sections and projections of the full space. Despite the fact that orbit types and bifurcations in high-dimensional spaces have not been completely classified, nonetheless it is still possible to recognize qualitatively different regions of behavior, and to map out these regions in state space and parameter space.

3.2 Computers, like people, need imagistic reasoning

In observing professional physicists and engineers, we are often struck by how an expert's "intuitive grasp" of a field is hard to articulate verbally. This is perhaps indicative of the use of non-verbal reasoning processes as part of the process of solving otherwise verbally presented problems. We observe scientists, mathematicians, and engineers continually using graphical representations to organize their thoughts about a problem. The programs we are developing use numerical methods as a means of shifting back and forth between symbolic and geometric methods of reasoning. The programs not only draw graphs and state-space diagrams, but they look at these diagrams and hold them in their "mind's eye" so that powerful visual mechanisms can be brought to bear on what otherwise would be purely symbolic problems.

The idea that problem solvers employing visual, analogue, or diagrammatic representations can be more effective than those relying on linguistic representations alone is not new. Even before 1960, Gelernter's Geometry-Theorem Proving Machine [10] used diagrams to filter goals generated by backward chaining. Nevins's [18] forward-chaining theorem prover focussed its forward deduction of facts on those lines explicitly drawn in a diagram. Stallman and Sussman's EL [23] program performed antecedent deductions in circuit analysis by exploiting the finite connectivity of devices.

What is provocative, however, is the suggestion that our thought processes are importantly imagistic, and that visual thinking may play a crucial role in problem solving. In scientific computation there has been tremendous emphasis on visualization, but this has mostly meant the development of computer-graphics technology to aid *human* visualization [17]. We believe

that imagistic reasoning is a very general class of problem-solving strategies, each with its own appropriate representations and technical support. The programs discussed above suggest that it may be at least as important for scientific computation to develop visualization aids for programs as well as for people.

References

[1] H. Abelson, M. Halfant, J. Katzenelson, and G.J. Sussman, "The Lisp Experience," *Annual Review of Computer Science*, 1988 (to appear).

[2] H. Abelson and G.J. Sussman, "The Dynamicist's Workbench I: Automatic Preparation of Numerical Experiments," MIT Artificial Intelligence Laboratory Memo no. 955, May 1987.

[3] M. Abramowitz and I. Stagun, *Handbook of Mathematical Functions*, Dover Publications, 1965.

[4] J. Applegate, M. Douglas, Y. Gürsel, P. Hunter, C. Seitz, G.J. Sussman. A digital orrery. *IEEE Trans. on Computers*, Sept. 1985.

[5] S. Baase, *Computer Algorithms*, Addison-Wesley, 1978.

[6] M. Feinberg, "Chemical oscillations, multiple equilibria, and reaction network structure," in *Dynamics and Modelling of Reactive Systems*, eds. W. Stewart, W. H. Ray and C. Conley, Academic Press, New York, 1980, pp. 59–130.

[7] V. Franceschini, "Two models of truncated Navier-Stokes equations on a two-dimensional torus," *Phys. Fluids*, vol. 26, no. 2, 1983, pp. 433–447.

[8] M. Halfant and G.J. Sussman, "Abstraction in Numerical Methods," MIT AI Memo 997, October 1987. To appear in proceedings of *ACM Conference on Lisp and Functional Programming*, 1988.

[9] M. Hénon, "Numerical Study of Quadratic Area-Preserving Mappings," *Quarterly Journal of Applied Mathematics*, vol. 27, 1969.

[10] H. Gelernter, "Realization of a geometry theorem proving machine," *Proc. Int. Conf. on Information Processing*, Paris: Unesco House, 1959, pp. 273-282; also in *Computers and Thought*, Feigenbaum, E. and Feldman, J. (eds.), McGraw-Hill, New York, 1963, pp. 134–152.

[11] J. Guckenheimer and P. Holmes, *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields*, Springer-Verlag, 1983.

[12] C.S. Hsu, "A theory of index for point mapping dynamical systems," *Journal of Applied Mechanics*, vol. 47, 1980, pp. 185–190.

[13] C.S. Hsu, W.H. Cheng, and H.C. Yee, "Steady-state response of a non-linear system under impulsive parametric excitation," *Journal of Sound and Vibration*, vol. 50, no. 1, 1977, pp. 95–116.

[14] H. Kinoshita and H. Nakai, "Motions of the Perihelions of Neptune and Pluto," *Celestial Mechanics*, vol. 34, 1984.

[15] K. Laidler, *Chemical Kinetics* (3rd edition), Harper & Row, New York, 1987.

[16] R. MacKay, *Renormalization in Area-Preserving Maps*, Ph.D. thesis, Princeton University, 1982.

[17] B. McCormick, T. Desanti, and M. Brown (ed.), "Visualization in Scientific Computing," *Computer Graphics*, vol. 21, no. 6, 1987.

[18] A. J. Nevins, "Plane Geometry theorem Proving using Forward Chaining," MIT Artificial Intelligence Laboratory, Memo. no. 303, January 1974.

[19] G. L. Roylance, "Expressing mathematical subroutines constructively," MIT AI Memo 999, November 1987. To appear in proceedings of *ACM Conference on Lisp and Functional Programming*, 1988.

[20] E. P. Sacks, "Hierarchical reasoning about inequalities," AAAI, 1987, pp. 649–654.

[21] E. P. Sacks, "Piecewise linear reasoning," AAAI, 1987, pp. 655–659.

[22] , E. P. Sacks, "Automatic Qualitative Analysis of Ordinary Differential Equations Using Piecewise Linear Approximations," MIT Laboratory for Computer Science, Technical Report number 416, March 1988.

[23] G. J. Sussman and R. M. Stallman, "Heuristic techniques in computer-aided circuit analysis," *IEEE Trans. on Circuits and Systems*, **CAS-22**, 1975, pp. 857–865.

[24] G. J. Sussman and J. Wisdom, "Numerical evidence that the motion of Pluto is chaotic," *Science*, (to appear). Also available as MIT Artificial Intelligence Laboratory Memo no. 1039, April 1988.

[25] J.M.T. Thompson, "Complex dynamics of compliant offshore structures," *Proc. Royal Soc. London A*, vol. 387, 1983, pp. 407–427.

[26] J.M.T. Thompson and H.B. Stewart, *Nonlinear Dynamics and Chaos*, Wiley, 1986.

[27] David A. Waltz, "Generating semantic descriptions from drawings of scenes with shadows," in *The Psychology of Computer Vision*, P.H. Winston, ed., McGraw-Hill, New York, 1985.

[28] Andrew P. Witkin, "Scale-Space Filtering", *IJCAI-83.*

[29] K. Yip, "Extracting qualitative dynamics from numerical experiments," AAAI 1987.

[30] K. Yip, "Generating global behaviors using deep knowledge of local dynamics," AAAI 1988.

[31] C.T. Zahn, "Graph-theoretical methods for detecting and describing Gestalt clusters", IEEE, *Trans. on Computers*, Vol C-20, January 1971.

[32] F. Zhao, *An $O(N)$ Algorithm for three-dimensional N-body Simulations*, Technical Report number 995, MIT Artificial Intelligence Laboratory, 1987.